Energy-Efficient In-Memory Database Computing

Wolfgang Lehner
Institute for System Architecture
Technische Universität Dresden
01062 Dresden, Germany
wolfgang.lehner@tu-dresden.de

***Abstract*—The efficient and flexible management of large datasets is one of the core requirements of modern business applications. Having access to consistent and up-to-date information is the foundation for operational, tactical, and strategic decision making. Within the last few years, the database community sparked a large number of extremely innovative research projects to push the envelope in the context of modern database system architectures. In this paper, we outline requirements and influencing factors to identify some of the hot research topics in database management systems. We argue that—even after 30 years of active database research—the time is right to rethink some of the core architectural principles and come up with novel approaches to meet the requirements of the next decades in data management. The sheer number of diverse and novel (e.g., scientific) application areas, the existence of modern hardware capabilities, and the need of large data centers to become more energy-efficient will be the drivers for database research in the years to come.**

I. Introduction

Data management is a core service for every business or scientific application. The data life cycle comprises different phases starting from understanding external data sources and integrating data into a common database schema. The life cycle continues with an exploitation phase by answering queries against a potentially very large database and closes with archiving activities to store data with respect to legal requirements and cost efficiency. While understanding the data and creating a common database schema is a challenging task from a modeling perspective, efficiently and flexibly storing and processing large datasets is the core requirement from a system architectural perspective. For the last 30 years, disk-centric systems based on commodity hardware exploiting only a minimal set of "regular" operating system services have reflected the state-of-the art. Within the last years however, this picture has dramatically changed due to several reasons, but especially due to significant developments in the hardware sector. This awareness to be more focused on special capabilities of the underlying system currently implies a huge impact on the research as well as on the commercial data management ecosystem [1], [2].

Within this paper, we will highlight some of these changes and accompanying challenges, pinpoint open issues, and outline some solutions as examples for current and ongoing database research activities. In a first step, we will summarize the key challenges for large-scale data management from an application perspective and the direction of low-level services provided by the operating system or underlying hardware. We will argue that the time has come to re-think data management architectures and demand "support" from the application as well as from the low-level services. We identify flexibility as the key concept to cope with functional as well as non-functional requirements where improved energy efficiency is one of the most dominating factors.

978-3-9815370-0-0/DATE13/©2013 EDAA

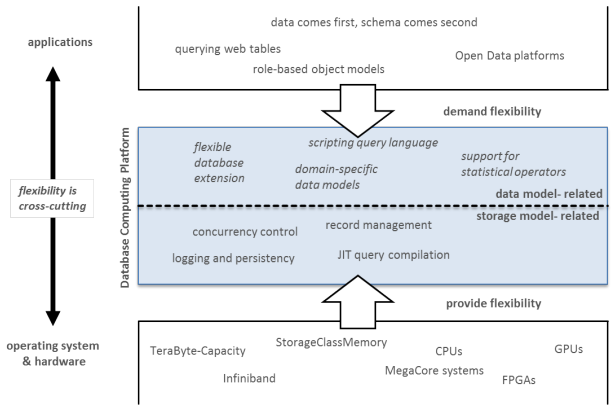

Fig. 1. General Situation of Database Management Systems.

II. Application Requirements

Currently, database systems are considered a software component providing a comprehensive SQL language layer and exploiting common services provided by the operating system and the hardware layer. However, current applications demand a quite different picture of a data management solution. As shown in Figure 1, data management functionality is challenged from two directions: From an application perspective and from a hardware and operating system perspective. From an application's point of view, we see the following major requirements:

- **Running a database system as a platform:** Classical database applications are based on application server infrastructures, which serve as an intermediate with respect to the database layer. Application logic runs within the application server layer providing services like load-balancing, session management, security framework, and distributed transactions to access multiple data sources. The price which has to be paid consists in SQL as

the only communication path between the data and the application. This results in multiple and extensive round-trips to express database interactions from an application layer perspective. Modern architectures have to provide a highly integrated platform of database and application server services within a single runtime to enable the application server component to directly access the internal service primitives of the database system.

- **Providing "data-as-a-service":** Database systems are usually running on-premise within a well-defined and well-administrated setup. Moving more and more services into cloud-like infrastructure with elasticity as one of the main driver, database systems have to learn how to act following the "as-a-service" paradigm. Although offerings at infrastructure-, platform-, or software-as-a-service exist in a wide variety, "data-as-a-service" is still in its infancy. This operational model requires substantial support from the system architectural side of a data management layer in order to natively support "elasticity in the large".
- **Support for flexible schema:** In modern, web-style applications scenarios, the schema is not known in advance but develops over time as data enters the system following the "data comes first, schema comes second"-paradigm. Therefore, the system has to constantly change the structure of the logical design with corresponding implications on the physical layer. As of now, schema changes reflect a major administrative task. Current database solutions require to carefully planning and designing the database schema, followed by filling the system with data, and finally performing necessary tuning operations like creating an index or partitioning the database to allow for partition pruning.
- **Coping with >10.000 tables in a query:** Traditional applications have dozens or a few hundred tables completely covering the semantics of a specific set of applications. Even in large application scenarios like ERP solutions with multiple thousands of tables (e.g., SAP ERP shows 50.000 tables) queries only referencing a small slice of the overall database (e.g., 10-100 tables within a query). Especially in web applications targeting knowledge extraction out of different web sources or open data platforms, 100s or even 1.000s of (weakly structured) tables within a single database query are common. Current compilation (especially optimization) components and database runtime infrastructures are not able to cope with this situation, forcing the application to split database interaction patterns into smaller pieces and merge the partial results within the application layer.
- **Providing hybrid query languages:** The SQL database language was originally intended for the application programmer. However, after more than 20 years of extending the language, SQL can only be generated by a software component and is no longer suitable for users like knowledge workers or data scientists, interactively working with the data. The original idea of declarative query languages consists in telling the system what to retrieve and not how to retrieve the required information and is still relevant. Additionally procedural elements are extremely worthwhile and should be part of a next generation data programming language. First ideas like JAQL [3], direct access to database elements via R [4], or the MapReduce programming paradigm [5] relying on 2nd order functions to provide an automated parallel execution of application logic are some examples of novel database languages.
- **Scaling to multiple billion record databases:** Database system are traditionally mostly used for transactional operations on small datasets, e.g., order processing, material distribution planning etc. However, more and more analytical applications are aiming at utilizing the mechanisms of a database management system. For example the transparent mapping of data model to storage model primitives by an sophisticated optimization framework is extremely beneficial for database applications. Referring to applications like sensor data or click-stream analysis, database systems have to cope with multiple billion record databases from a query processing perspective (e.g., exploiting massive parallelism) as well as from a data life cycle management perspective (e.g., aging data with respect to a given storage hierarchy or securing archived data from a posteriori modifications due to legal constraints).

III. Hardware and OS-level Requirements

While the data management layer is faced with a huge variety of requirements from the application side, recent and future developments at the hardware side and operating systems layer will additionally have a significant impact on the design of future database systems. However, in opposite to the requirements from the application side, the database community does consider these developments at the lower layers of the software stack as extremely valuable and beneficial in order to "outsource" some of the original core service primitives to the underlying software and hardware levels.

- **Enhanced synchronization methods:** While database systems provide concurrency control at the level of database objects, i.e., on table or tuple level, many of the internal data structures are based on traditional synchronization methods like locks and latches. In order to cope with a large number of computing units, more elaborate synchronization methods are highly welcomed by data-intensive tasks. For example, splitting an aggregation operator to compute the sales behavior of different customer groups into hundreds of different threads eventually implies high synchronization overhead, because every data stream may have database entries of different customer groups. Even read-only synchronization (to prevent from concurrent writers) already shows a significant serial part dramatically reducing the speedup with a growing number of parallel operators [6]. Optimistic concurrency

control mechanisms like Intel's TSX outlined in [7] are a big step forward.

- **Comprehensive xPU and co-processor support:** A significant number of research activities maps traditional database operators to non-standard hardware platforms, especially GPUs and FPGAs. As of now, only a limited number of operators show significant benefit when running on non-CPU hardware platforms. However, research activities are required to look into more complex and non-traditional database operators to support application scenarios beyond transaction processing or simple analytical aggregation scans. For example, [8] is looking into the efficient computation of frequent item sets based on a recorded set of transaction data. [9] is giving a wealth of other database-related problems for GPU usage.
- **Large-scale main-memory management:** Commodity servers equipped with 1 TByte main memory are currently available for a very reasonable price. While most of the servers follow the NUMA-architecture principles with local but cache-coherent memory layout, modern database systems exactly have to know the allocation scheme of the data in order to compute an optimal schedule for the operators of a given query. As a requirement, cache coherency should not always automatically be ensured at the hardware level, if the database system exactly knows the allocation scheme and the dependency between the different data sets. Additionally, modern server infrastructures like the prototype developed with the HAEC research project [10] deploys high-bandwidth, short-range wireless and optical links to dynamically configure the topology of the computer during runtime.
- **Multi-level reliability:** Depending on the semantics of a piece of data, different reliability constraints should be attached to a memory fragment. For example, intermediate results of a currently running query could be placed in some "cheap" memory with high write and read performance. On the other hand, REDO-log information, containing data of the least successfully committed transactions, should be stored in a replicated way, within a compute cluster or even across multiple locations. The database system therefore requires mechanisms to convey quality-of-service information about specific memory fragments to the underling software respectively hardware layer. The system can then decide for the most optimal way to achieve the required service-level request either in the fastest or cheapest way.

To put it into a nutshell, application demands and hardware developments suggest to completely rethinking the architecture of database management systems. While current state-of-the-art systems follow the classical assumption of being able to exploit a generic infrastructure and provide some extent of SQL for the application, novel architectural approaches have to architected around the cross-cutting concept of **FLEXIBILITY**. The overall hypothesis is that only a flexible system design comprising the overall system development, the query compilation framework, and the query runtime is able to cope with current application challenges. Also, we think that flexibility is the core to achieve energy efficiency without compromising the overall system performance. We will detail some of our thoughts in the following section.

IV. Consequences for Research Activities

As already outlined, a huge number of functional extensions is necessary to satisfy the already existing demand of non-standard database applications. However, in addition to the functional extensions, we see four major goals that have to be achieved from a non-functional perspective:

- **Performance:** Following the famous statement of Bruce Lindsay [11], performance is the most relevant non-functional property of a data management platform. High performance is and will remain the main driver for future database systems. However, we see application domains–especially in the context of scientific computing–where throughput optimization is more important than response time optimization of a single query. Query optimization frameworks have to consider this shift of the overall optimization goal, which is also highly correlated to improved energy efficiency (see below).
- **Energy efficiency:** In order to improve the energy efficiency of a database server, two independent directions can be pursued. On the on hand, the faster a query is being processed, the less energy is consumed [12]. For example, if a query can be answered using an index lookup instead of a table scan, fewer cycles are spend on that particular query. In that respect, traditional query optimization is still implicitly an optimization with respect to energy efficiency. However, those naive considerations fail, if queries are executed in a distributed environment with additional communication costs. For example, an optimizer has to decide about sending intermediate data in a compressed or uncompressed format to other nodes or even sockets on the same board. In the former case, the system has to spend time and energy for (de-)compression but saves time and energy for the communication path. Since both cost factors are independent, the optimizer has to decide on a case-by-case basis. On the other hand, energy can be saved, if individual hardware components are turned off to save idle power and increase the utilization of running components. As a consequence, the individual response time of a query may suffer from improved energy efficiency. Again, the system has to flexibly balance query response time minimization and throughput maximization under a given energy constraint on a case-by-case basis (Figure 2). In the opposite to "elasticity in the large" (section III), this property can be considered "elasticity in the small".
- **Robustness:** Current query and transaction execution semantics force a database request to abort and roll back if one single failure occurs, be it on a node level or within an individual operator. One of the lesson learned from Hadoop-like infrastructures, a future database system

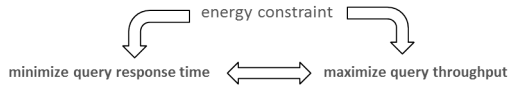

Fig. 2. Impact of Energy Constraint on Query Optimization.

should in a much wider sense compensate for failures and hide failures with respect to the application. The effort should be balanced with respect to the individual queries: while short read requests can be easily repeated, intermediate results of long-running analytical queries or long-running transactions have to be preserved and transparently used for a restart.

- **Reliability:** While robustness focuses on an individual user interaction with the system, reliability addresses the overall system stability and physical consistency of the persistent database. While more and more features are added to a database platform, principles of separation of concern have to be the overall guideline for system development.

The central question is: How to achieve those functional and non-functional optimizations beyond flexibility during compile and runtime? The answer is two-fold: On the one side, the application layer has to acknowledge some relaxations in terms of the expected services potentially shaking some of the classical data management principles. On the other side, significant research investment is needed to significantly better exploit existing solutions provided by the hardware and to demand even more sophisticated low-level services. We will comment on these statements in more detail.

A. Assistance from the Applications

Applications typically require all the "nice" services of database systems at no price: transactional guarantees (preferably strict serializability), high availability of the system, automatic physical design of the database, seamless integration into the application programming philosophy. However, as we have seen in the context of NoSQL stores, some restrictions have to be accepted by the application to be able to fulfill the wish list of a scalable data management solution. For example, Key-values stores reduce the schema to only the key/value-pair. No additional constraint checking, no complex processing framework etc. is required; parallelization can be automatically performed on the key-level without any side-effects. On the downside however, the application has to take care of the specific semantics of the "value" and decompose the "value" of the database into application-specific entities. A second example can be seen in the BASE consistency model [13]. Changes to the database will eventually make it to all replicas in a distributed system, but there is no serializability guarantee given to the application. The application itself has to ensure that–for example–two objects read from the same database are actually representing the same state with respect to concurrently running updates. Following this line of thought, more alternative concepts could be envisioned to gain some support by future database platforms, e.g.:

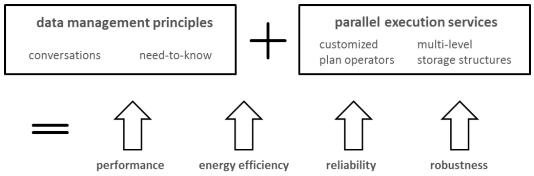

Fig. 3. Research Potential for Database Computing.

- **Database conversations:** In addition to the traditional transactional model, database conversations may help to free the database system from managing and maintaining the single point of truth. The concept of database conversations creates application specific views on top of the underlying database which are materialized (i.e., exist beyond the scope of a single application transactions) and can be shared with others. The "community" of applications are creating potentially different domain-specific versions of the original database in a step-by-step manner; there is no need for the database itself to ensure correctness with respect to all potential applications.
- **Need-to-Know principle:** The Need-to-Know principle states that the system has to reflect only that degree of consistency, which is required by a specific application. In the opposite, the traditional principle of ubiquity requires from the system a consistent state at any point in time without considering any users. For example, a system following the principle of ubiquity has to maintain an index entry after an update in the database independent of any reader referring to the index. A system following the Need-to-Know principle would only update the index if another application has indicated interest in reading the index.

B. Assistance from the Hardware Side

Forcing the application to take over some of the burden of the database system creates extensive degree of freedom, which could be exploited by the architectural design. However, these opportunities coming from the application side have to be supported by an efficient storage and query execution engine able to orchestrate a huge number of parallel tasks. Parallelism in general can be positioned as the core principle to achieve high performance on the one hand and to enable energy efficient processing schemes with an appropriate scheduling design on the other hand. Parallelism has to be considered in an end-to-end manner starting from the query language level down to the execution runtime. Particularly, we see research potential in the following areas:

- **Customized plan operators:** Recent developments focus on efficient code generation as an alternative to build a data-flow graph based on pre-compiled plan operators [14]. However, in addition to the compilation process, next generation operators should be "hybrid" and "reconfigurable" [15]. The first property targets heterogeneous hardware components. As of now, an operator (or even a complete query) may be deployed either on a CPU or

on a GPU platform, e.g. [16]. Next generation operators are supposed to exploit an even more fine-grained distinction. For example, while init() and finish()-phases of operators may run on a CPU side, the actual work()-part of an operator may be scheduled on a GPU platform. Orthogonal, operators have to quickly adapt to changing data characteristics and potentially to changing hardware structures. For example, selectivity factors significantly impact the success of branch prediction forcing the operator to switch between different implementations [17]. Even more reconfigurability is required, if the hardware can be restructured at runtime, e.g., trading cores against cache capacity.

- **Multi-level storage structures:** Within the last years, a significant shift from disc-centric to main memory-centric systems can be observed. Large main memory capacities are economically affordable and many database scenarios fit into main-memories of today's commodity servers. This development can be considered a substantial shift in database architectures: On the one hand, the complete storage hierarchy shifted "one level up"; main memory is the new disk, disk is the new archive. Interestingly main-memory and disk share some common characteristics. For example, both techniques are block-oriented, where cache lines may be considered the new block size and the CPU cache management may reflect the new buffer manager. On the other hand, main memory-centric database system design denotes radical change: For example, as shown in [18], novel concurrency schemes are heavily relying on direct access to the database objects without any significant performance penalty. With the advent of persistent main memory, even more advanced techniques in the context of logging will appear [19]. Nevertheless, disks will still play a major role in large database installations. Physical database design will distinguish between "low-density" and "high-density" data. High-density data like order entries or other business-critical objects with high transaction load will stay and manipulated in main-memory. "Low-density" data represents primarily read-only data coming from sensors or web activities (click streams) and will be placed on traditional cheap disk devices; low-density data in general does not have any semantics per se but is used for statistical hypothesis testing. While point access is typical for high-density data, low-density data is usually queried by massive and parallel scans against large disk-farms. Moving data between different levels in a storage hierarchy is currently under the control of the database system but could be extremely well handled by system-level services.

V. Summary

Flexibility is the key for future database system architectures. On the one hand, novel applications are driving database technology requesting performance and adaptivity with respect to the web-style data processing, large volumes, short load-to-query times etc. On the other hand, recent developments in the hardware sector require a total rewrite of database systems. Large main memories and a high number of cores are the main drivers as of now; persistent memory, advanced synchronization, reconfigurable hardware etc will be the driving factors of the future. Improved energy efficiency–or more specifically query processing under a given energy constraint–will be a constant challenge which has to be tackled from many different directions during compile and runtime. More generally speaking, database systems are and will be one of the most demanding but also grateful "applications" for advances in hardware and operating systems.

VI. Acknowledgments

This work is partly supported by the German Research Foundation (DFG) within the Collaborative Research Center 912 "Highly Adaptive Energy-Efficient Computing".

References

[1] H. Plattner, "A common database approach for oltp and olap using an in-memory column database," in *SIGMOD Conference*, 2009, pp. 1–2.

[2] M. Stonebraker, "Technical perspective - one size fits all: an idea whose time has come and gone," *Commun. ACM*, vol. 51, no. 12, p. 76, 2008.

[3] K. S. Beyer, V. Ercegovac, R. Gemulla, A. Balmin, M. Y. Eltabakh, C.-C. Kanne, F. Özcan, and E. J. Shekita, "Jaql: A scripting language for large scale semistructured data analysis," *PVLDB*, vol. 4, no. 12, pp. 1272–1283, 2011.

[4] P. Große, W. Lehner, T. Weichert, F. Färber, and W.-S. Li, "Bridging two worlds with rice integrating r into the sap in-memory computing engine," *PVLDB*, vol. 4, no. 12, pp. 1307–1317, 2011.

[5] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in *OSDI*, 2004, pp. 137–150.

[6] R. Johnson, I. Pandis, N. Hardavellas, A. Ailamaki, and B. Falsafi, "Shore-mt: a scalable storage manager for the multicore era," in *EDBT*, 2009, pp. 24–35.

[7] J. Reinders. (2012, Nov.) Transactional synchronization in haswell. [Online]. Available: http://software.intel.com/en-us/blogs/2012/02/07/transactional-synchronization-in-haswell

[8] B. Schlegel, R. Gemulla, and W. Lehner, "Memory-efficient frequent-itemset mining," in *EDBT*, 2011, pp. 461–472.

[9] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krüger, A. E. Lefohn, and T. Purcell, "A survey of general-purpose computation on graphics hardware," 2007.

[10] W. Lehner, "Pathways to servers of the future," *IEEE Technology Time Machine*, 2012.

[11] M. Winslett, "Bruce lindsay speaks out: on system r, benchmarking, life as an ibm fellow, the power of dbas in the old days, why performance still matters, heisenbugs, why he still writes code, singing pigs, and more," *SIGMOD Record*, vol. 34, no. 2, pp. 71–79, 2005.

[12] D. Tsirogiannis, S. Harizopoulos, and M. A. Shah, "Analyzing the energy efficiency of a database server," in *SIGMOD Conference*, 2010, pp. 231–242.

[13] W. Vogels, "Eventually consistent," *ACM Queue*, vol. 6, no. 6, pp. 14–19, 2008.

[14] T. Neumann, "Efficiently compiling efficient query plans for modern hardware," *PVLDB*, vol. 4, no. 9, pp. 539–550, 2011.

[15] T. Kissinger, B. Schlegel, D. Habich, and W. Lehner, "Qppt: Query processing on prefix trees," in *CIDR Conference*, 2013.

[16] C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. A. Brandt, and P. Dubey, "Fast: fast architecture sensitive tree search on modern cpus and gpus," in *SIGMOD Conference*, 2010, pp. 339–350.

[17] K. A. Ross, "Selection conditions in main memory," *ACM Trans. Database Syst.*, vol. 29, pp. 132–161, 2004.

[18] P.-Å. Larson, S. Blanas, C. Diaconu, C. Freedman, J. M. Patel, and M. Zwilling, "High-performance concurrency control mechanisms for main-memory databases," *CoRR*, vol. abs/1201.0228, 2012.

[19] R. Fang, H.-I. Hsiao, B. He, C. Mohan, and Y. Wang, "High performance database logging using storage class memory," in *ICDE*, 2011, pp. 1221–1231.